

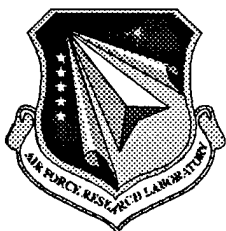
DEVELOPMENT OF VISUALIZATION MODULES FOR ICEPIC

Kimberly Robinson

August 1998

Final Report

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.



**AIR FORCE RESEARCH LABORATORY
Directed Energy Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776**

DTIC QUALITY INSPECTED 4 1 9990416029

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data, does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report has been reviewed by the Public Affairs Office and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

If you change your address, wish to be removed from this mailing list, or your organization no longer employs the addressee, please notify AFRL/DE, 3550 Aberdeen Ave SE, Kirtland AFB, NM 87117-5776.

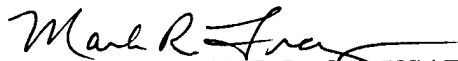
Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

This report has been approved for publication.



GERALD E. SASSER III, Capt, USAF
Computational Physicist

FOR THE COMMANDER



MARK R. FRANZ, Lt Col, USAF
Chief, High Power Microwave Division



R. EARL GOOD, SES
Director, Directed Energy Directorate

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 18 Aug 98	3. REPORT TYPE AND DATES COVERED Final Report, Jun-Aug 98		
4. TITLE AND SUBTITLE Development of Visualization Modules for ICEPIC		5. FUNDING NUMBERS		
6. AUTHOR(S) Kimberly Robinson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFRL/DEHE 3550 Aberdeen Ave Kirtland AFB, NM 87117		8. PERFORMING ORGANIZATION REPORT NUMBER AFRL-DE-PS-TR-1998-1060		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING / MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (<i>Maximum 200 Words</i>) <p>AVS/Express Developer's edition, by Advanced Visual Systems Inc, is a complex data visualization kit consisting of several hundred different modules, as well as a couple different editors, each serving a somewhat different function. In order to use this program, extensive training is required. In hopes of making things easier by designing a system with which people not familiar with the specifics of AVS can create visuals fairly quickly, an alternate method was developed. Through the custom designing of new modules specified to do certain things simply, graphs and movies are now considerably easier to make and include their own, easy to use, interfaces. This assists the computationalists at the Center for Plasma Theory and Computation by allowing them to view the results of their simulations more easily.</p>				
14. SUBJECT TERMS Plasma Physics AVS/EXPRESS			15. NUMBER OF PAGES 22	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited	

Table of Contents

List of Figures	iv
List of Tables	iv
Introduction	1
AVS Overview.....	1
Tools and Methods.....	3
Converting ICEPIC data to AVS field format.....	5
Conclusion	14
Acknowledgements.....	14
References	15

List of Figures

FIGURE 1: EXAMPLE OF LIBRARY WORKSPACES	3
FIGURE 2: THE READ MESH NETWORK	6
FIGURE 3: RADIAL ELECTRIC FIELD IN THE MAGNETICALLY INSULATED LINE OSCILLATOR	7
FIGURE 4: DATAMAP CONTROL PANEL	8
FIGURE 5: UI PANEL	9
FIGURE 6: WRITE_IMAGE PLUS STRING	11

List of Tables

TABLE 1. THE COLOR OF THE LINES CONNECTING MODULES CORRESPONDS TO THE TYPES OF DATA WHICH MAY BE TRANSMITTED ALONG THEM.	2
---	---

Introduction

The computationalists at the Center for Plasma Theory and Computation (CPTC) run simulations with plasma physics software that create very complicated data. The easiest way of understanding this data is through the use of images created directly from the data. For this purpose, they purchased a software package called AVS/EXPRESS produced by AVS (Advanced Visual Systems), Inc [1]. It is multi-platform but runs very well on a Silicon Graphics machine due to the advanced graphics rendering hardware on that machine. Unfortunately, the learning curve associated with this software is fairly steep, so that a significant commitment of time is required to make even simple graphics. In order to significantly reduce the time required for members of the CPTC team to be able to create useful graphics, graphical interfaces and custom modules were developed that were specially tailored to the data created by the CPTC software packages. The process of creating specialized modules for the scientists took a bit of time, some C-coding, and a bit of creativity. Now there are modules that work exactly how they are needed to and contain specific interfaces that negate the need for one to understand the details of how AVS works. In this paper terms that are specific to AVS modules or pathnames will be Arial font while the underlined Arial represents modules that have been built, and terms that have a unique meaning in AVS will be in *italics*.

AVS Overview

The AVS software is based on groups of objects called modules. These modules often deal with data formatted in the AVS *field* format, which consist of a *mesh* combined with cell data. The AVS *fields* are not always the most convenient format to use. A *field* file can be read into AVS with the use of the AVS Read_Field module, but it has to be in a very specific format that involves a good deal of data post-processing before it can enter AVS. There are several other methods of reading in data that work better but most are still difficult to work with. The simplest of these is through the creation of a File_Import module. File_Import modules may be created and customized to the format of the files that are to be read in. Such modules read in multiple columns of data and export them as arrays. They can also be modified for specific files.

To build a `File_Import` module, a `Workspace` in the `User_Workspaces` library must be selected (see Figure 1). Usually, the `User` space should be reserved for modules that include user-provided source code. Under the `Object` menu on the menu bar is `Add File_Import_Module`. This opens an interface that prompts the user through the process, which includes naming the module, adding variables and assigning those variables to columns of data. It also wants to know if the file is ASCII or binary, which does make a difference. There is a bug in the ASCII reader: it prompts for how many bytes to skip, which does not work, so header lines do not work in ASCII files. It should be noted that these modules will only have the function of reading in data (user specified source code cannot be included.)

Data are passed and modified in AVS through a series of modules in a *network*. After the addition of several modules, this looks like a tangle of colored spaghetti. Each module has colored ports of what it can take in or put out. Modules are connected through colored tubes that eventually get tangled and provide a confused mess to work with. In general, the colors mean specific things, but depending on the module, a color can mean something different; black, for instance, can be several different things. Usually the colors correspond to the types of data as shown in Table 1.

Table 1. The color of the lines connecting modules corresponds to the types of data which may be transmitted along them.

Type of data	Color	Example
Renderable image	Red	<i>Mesh</i> output to viewer.
Field	Blue and Black stripes; All black;	Output of <i>mesh</i> to input of data module (orthoslice).
Primitive data type	Blue, Black, green stripes All black (single values, not <i>fields</i>)	
Float data type	Brown	Arrays of floats come from <code>File_Import</code> modules.
Int data type	Pink	
String data type	Blue	
Parent connection	Bright green	
UI connections	Pastels	Used primarily in UI

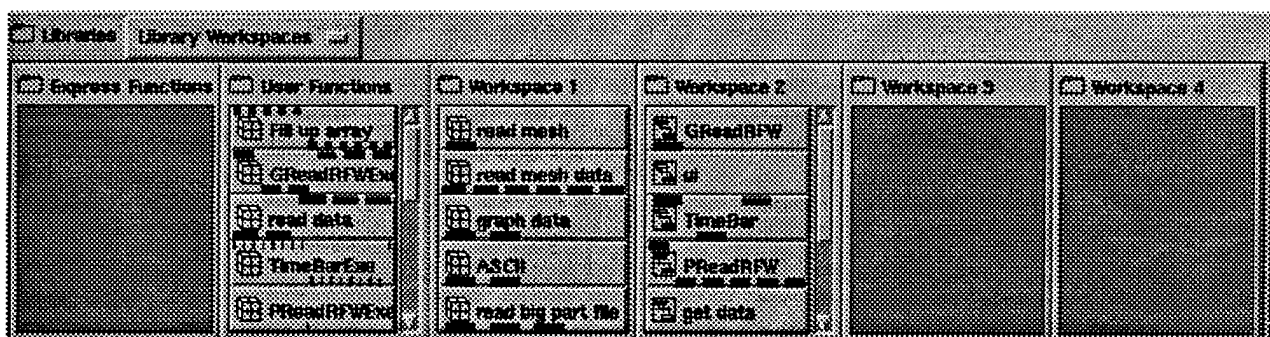


Figure 1: Example of Library Workspaces

Tools and Methods

AVS Applications are built in the *network editor* (An example is provided later in Figure 2).

Typically, the first item in an AVS *network* is a module that is used to read in the data file. This can be hard to do for someone unfamiliar with AVS. As described above, the `File_Import_Module` is the simplest way of reading in a file with data in columns. The next step is to do something with that data. Most applications will require a *mesh* to be built; it is similar to a *field* and will often work the same way. A *mesh* may be created by inputting coordinate information into the appropriate module found under `Main.Field_Mappers.Mesh_Mappers`. There are several types of *meshes* that can be used, but all are fairly straightforward. A point *mesh* tends to work best for plotting scattered particles. Where to go from here depends on the type of plot desired. AVS can make many different kinds of plots, all requiring different kinds of input. Images that are of particular interest to researchers at CPTC include contours of electric and magnetic fields, and the movement of particles through complex geometric shapes. Those plots will be focused on, since they were the ones developed.

Several tools were developed to visualize data produced by the plasma physics simulation software, ICEPIC [2]. ICEPIC is a particle-in-cell (PIC) code that was developed at the Air Force Research Laboratory (AFRL) primarily to assist in the development of high power microwave sources. ICEPIC produces a variety of data sets: each in independent files. These files include representations of the device geometry, electric and magnetic field information for all of the computational domain, charged particle positions and velocities, electrical current, transmitted power, etc. One of the projects accomplished this summer was a movie, which showed several of these data sets at once and their evolution with time. This allowed users to see the interaction of the data sets, giving them a "big picture" view of the simulation.

A picture of the geometry may be generated by viewing a quads output file. This file contains a series of quadrilaterals (four points in a plane) which are created by checking all of the faces of each cell within the problem and outputting those faces which lie on a physical boundary. The quads file is reduced to a workable size by a C code, quadreduce, written by Jerry Sasser of CPTC. It is read in using a macro (a collection of modules) called GReadRFW built by Kent Eschenberg from the Army Corps of Engineers Waterways Experiment Station (CEWES) Major Shared Resource Center (MSRC) in Vicksburg, Mississippi. GReadRFW includes a module built from C code with the Add_Module function that reads the geometry file and puts it into a point *mesh* format.

A PIC code tracks charged particles as they are moved through a computational mesh under the force of electric and magnetic fields. It is often interesting to view the location of these particles with respect to the device boundaries. Dr. Eschenberg also wrote a C program that makes this process much more efficient. Due to the number of particles that appear at each subsequent time step, around 1×10^6 , it becomes very time-consuming to read them in as point *meshes* one step at a time, so that it has to read and map every particle over and over again. The RF Particle Counter (rfpc) pre-processor breaks the problem domain into a $256 \times 256 \times 256$ mesh of boolean data points. Each point is set to 1 if at least one of the particles exists in that region and 0 if none do. This array of booleans is then created for each time step and stored in a new file, which takes up considerably less space than the original data files. The rfpc program allows the Loop, which controls the movie, to run much faster. Those two modules are the basis of the development that is covered here.

Another image that was produced was a 2-dimensional contour plot of the radial electric fields in a cross-sectional cut through the volume. This technique is easily modified to view any component of the electric or magnetic field. This process is more complicated because the modules involved require an AVS *field* as input (in particular, one with a rectilinear or uniform *mesh*.) The electric field data output by ICEPIC is not compatible with the AVS *field* format and requires significant postprocessing. To understand the format of the electric field data consider the following example. The computational space in ICEPIC forms a parallelepiped (like a shoebox). This space is broken into rectangular cells by defining a grid as the intersection of an x-mesh with a y-mesh and a z-mesh, where each mesh is a 1D array that describes how that direction is to be divided. Now suppose that the device of interest is actually a cylinder

which fits within the computational space. ICEPIC only performs computations on cells which fall within the physical domain (in this case the cylinder, which has stair-stepped edges since the cells are rectangular) and no memory is allocated for the other cells. When ICEPIC outputs the electric field data, it goes through its list of real cells and prints the cell position (x, y, z) and then the components of the electric field (E_x , E_y , E_z). This is very different from the *field* format which would expect all of the x-mesh values, all of the y-mesh values, all of the z-mesh values, and then electric field values for every cell (including those outside the cylinder since AVS is not aware of its existence) in a particular order. Since reading in the files as a `File_Import_Module` was really too simple a format in this case, something else had to be created. It was necessary to build a new module, from scratch. This is similar to building a `File_Import_Module`: a library, preferably `User`, must be selected and the command (`Add_Module`) is found under the object menu. The interface prompts for variables and their roles in the module, but then it allows for the insertion of source code. Before generating source code, it is best to allow the new module interface to create a source code template given the desired inputs to and outputs from the module. This is done by clicking on *edit source* in the last window of the `Add_Module` tool before a file of the expected name exists. The template will contain all of the necessary Application Program Interfaces (APIs) required to communicate with AVS. In creating the module it is important to think ahead and carefully design the module. It is more difficult to add new APIs after the template has been created. These modules can be written in C, C++, or Fortran; the language option is the first thing that appears on opening the build module interface.

Converting ICEPIC data to AVS field format

As mentioned above, there are two steps involved in creating an AVS *field*: generating a *mesh* and assigning data to the *mesh*. The meshes that come from ICEPIC have to undergo several changes to get into AVS. Three files, *xmesh*, *ymesh*, and *zmesh* combine to make a range of values which may be contained in the ICEPIC field data file. Those are read into the *network* (see Figure 2) with three separate `File_Import_Modules` named `read_mesh`. Each array is interleaved with itself with the `interleave_3_arrays` module from `Main.Mappers.Combiners`. Finally, those three arrays are catenated to form an $[n_x+n_y+n_z][3]$ array of points that define the *mesh* that will be used in the AVS *field*. This is done with the `concat_3_arrays` module, which is found in the same menu as the `interleave` module. It does not matter if the three *mesh* dimensions are different because the interleaving takes care of that.

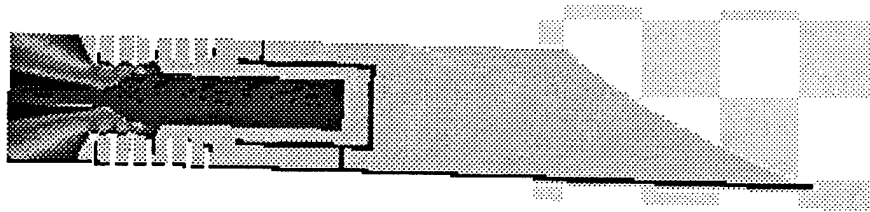


Figure 3: Radial electric field in the Magnetically Insulated Line Oscillator

Before the data are inserted in the large data array, the elements of the array are initialized to a value that is larger than any of the data (this value is referred to as the null value). This will be used to take advantage of several useful AVS filter modules. The AVS *field* is created by piping the mesh array and the data array into a `Rect_Scalar_field` module, which creates a *field* consisting of a rectilinear *mesh* and scalar data. This module also requires `dims`, which is an array of the three mesh sizes. `Points` is the 2D array of the *mesh* points. Data is what came out of the `Fill_up_array` module. In this case it should be scalar, so E_x was chosen.

The `Rect_scalar_field` module outputs a field that can be fed into a module called `orthoslice`, under `Mappers`. `Orthoslice` shows a 2D plane of data in a cross-section which may be chosen to be in any of the 3 directions and may cut along any *mesh* point. If the shape of the object being simulated is not rectangular (for example, the cylinder discussed earlier), there will likely be null values surrounding the data that is wanted. These can be removed by using the `threshold` module, under `Main.Filters`. This was the purpose of the pre-initialization of the data array discussed above. When the null value is set everything with that value will disappear. The output of `threshold` feeds into `orthoslice` and `orthoslice` to the `Uviewer` (using the red pipe). The `Uviewer` should show a 2D plane colored in the shape of the device. This can be displayed over the device geometry by adding a `GReadRFW` module. Remember that when working with multiple images care must be taken not to edit the size or position of one of the images if it is important that they have a fixed position. This lesson is easily learned the first time a project has to be restarted after accidentally editing only one object's size or position. The results of an `orthoslice` and an image of a device are shown in Figure 3, where this is an early time step for the Magnetically Insulated Line Oscillator (MILO) [3].

The colors chosen for the contours are specified in the datamap. The colors are scaled to the data and if the null value used to preinitialize the electric field array is much larger than the real data, the image

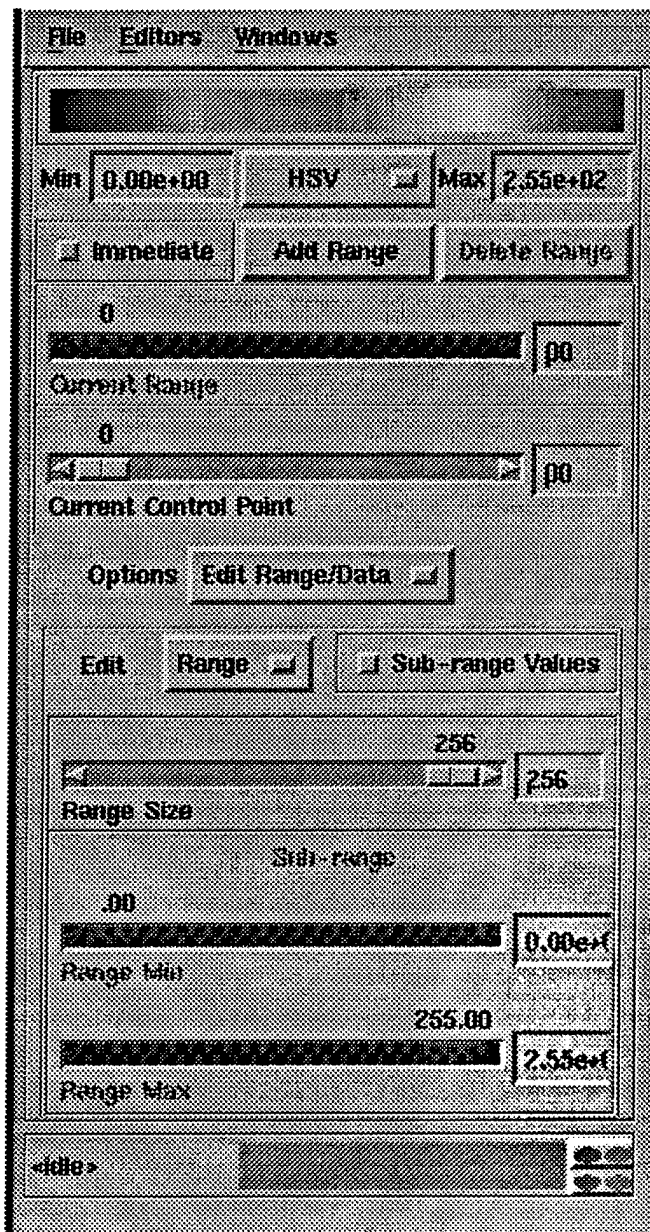


Figure 4: Datamap control panel

will be shaded entirely in one color. This can be taken care of but should be done carefully. It is not a good idea to delete the data map. There is a button at the bottom of the Uviewer window entitled "select object". It brings up a menu of the modules that are linked directly to the viewer. This makes it possible to change the properties of individual objects. It also makes it possible to destroy the project fairly easily if something is moved separately when it should not be. Select orthoslice in the object menu and datamap editor in the editors menu on the menu bar. The datamap appears in the viewer interface window (See Figure 4). The next modification depends on where the data are in the range. If the data consist of small values, on the lower end of the, an additional range will work well; select Add_Range. This gives the whole range of

the current map to the colors in the reduced data range. There is a pop-up sub-menu in the middle of the editor box. One of those options is Edit Range/Data. It provides an interface that includes a sub-range. Toggle the box that gives a sub range. It has values from the file it read. Play with the min and max of those values until they correspond to the

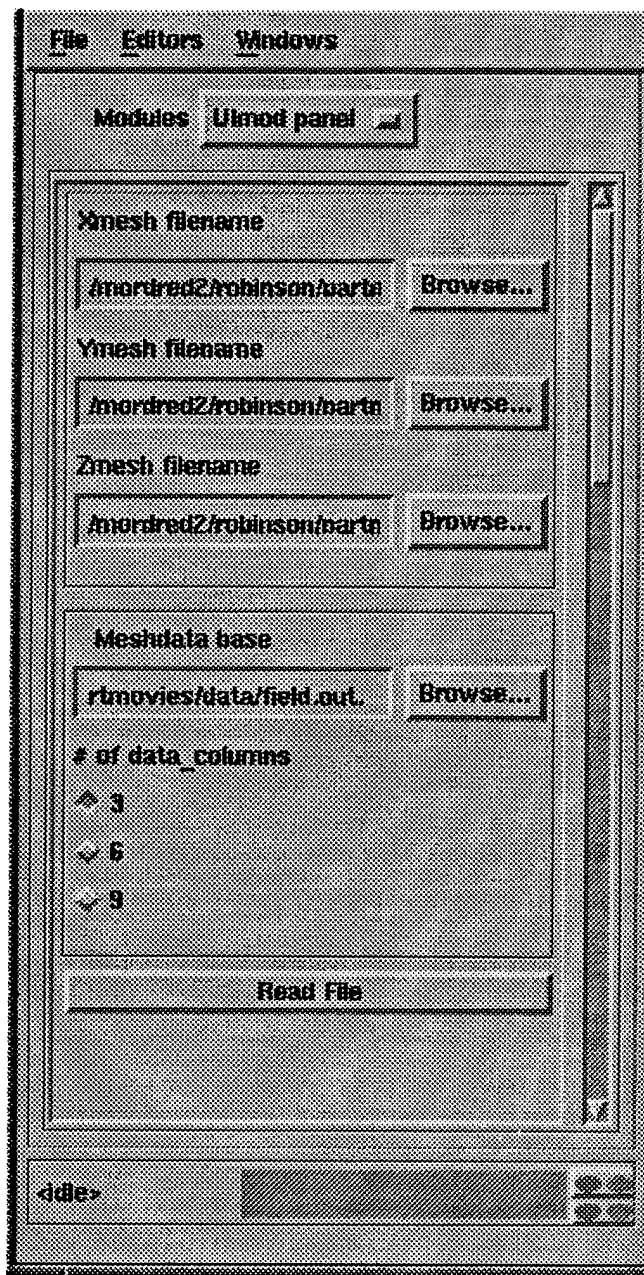


Figure 5: UI panel

object menu can also change the color of the object and the jitter, which layers things in the viewer, allowing things to be seen or not seen through another object (this is often useful when working with particles).

In order to make the modules more user-friendly, user interfaces were added using the AVS User Interface (UI) kit. This kit allows the user to create an interface similar to those on the AVS modules. It provides a series of panels, buttons, sliders, and type-in areas to get the information to the

range of the real data. If the data are toward the middle of the range, add a sub-range, but not an extra range. Another option is to change the colors with the Edit Color sub-menu. Control point 0 (the slider above the menu) represents the data min while control point 1 represents the color at the data max. If the 256 range does not work for whatever reason, under Edit Range/Data is a slider that will reduce the colors in the range.

A very nice trick that allows other objects (such as the device geometry) to be seen through the slice may be found in the Object editor, under the Editors menu in the Uviewer. The orthoslice must still be selected. Under Properties.Surface are several sliders that control the surface properties of the object. One of these is opacity; reducing this makes the slice somewhat transparent so that it seems to be

more a part of the geometry. The

modules that require it. Each module that was created for this project has its own interface so that it can be used with very little knowledge of AVS. For example, Figure 5 shows the UI for the macro which reads the electric field data.

Quadrilateral and contour plots have already been discussed, but there are many more visualization methods available in AVS. Many of the 3D plots can be found under **Main.Mappers**. The annotation and graphing kit takes care of the 2D graphs. **AGGraph** creates a 2D line plot based on an array of y values. X values can be provided but are not necessary. **AGGraph** comes in various formats (cartesian, polar, bar charts, etc.) along with all sorts of axes and legends. There are some difficulties in combining an **AGGraph** with 3D images. Whenever a 2D graph is put in the same viewer with a 3D object, it blinks and then disappears. It turns out that this is a problem with the hardware rendering. It works fine with the **Uviewer** that supports 2D and 3D and with software rendering toggled.

An example of an application requiring the 2D graph is the changing power output of the device. Making this graph change in time (along with the other visuals) provided some unique challenges. The goal was to make it loop through the array of data so that it would be animated, presenting a problem that could not be fixed with the existing modules. The module that was developed used the count variable from the loop module to set the size of the array that would be read in. By giving the graph module the entire x array, but only the partial y array (up to the loop counter), it drew the graph as the loop incremented giving the impression of an animation. There is also a string variable that allows the color of the graph to be changed.

Another module that Dr. Eschenberg built is a **TimeBar** that represents the location (i.e. the time in the simulation) in the loop by changing the color of a bar from gray to red with the option of red to green at a certain value as it moves along. A small arrow moves ahead of the red section according to the loop count.

Lights, Camera, Action: Making movies with AVS

There are two ways of making a movie with AVS. One involves creating images in AVS one at a time and saving them as gif or jpg files and then using additional software to catenate the images into a movie. The other came built into the most recent edition of AVS. It does not work quite as well for data

movies. It is really more for making movies of alterations to the view of the data set, such as a fade in or a rotation rather than altering the data itself. Things to consider before saving the images include lighting, which will not be as good in the image as it appears on the screen, so consider adding more light. The positioning needs to be correct as well; it is sometimes better to rotate and translate objects using the Transform editor (in the editors menu of the viewer) rather than moving them with the mouse. If there is

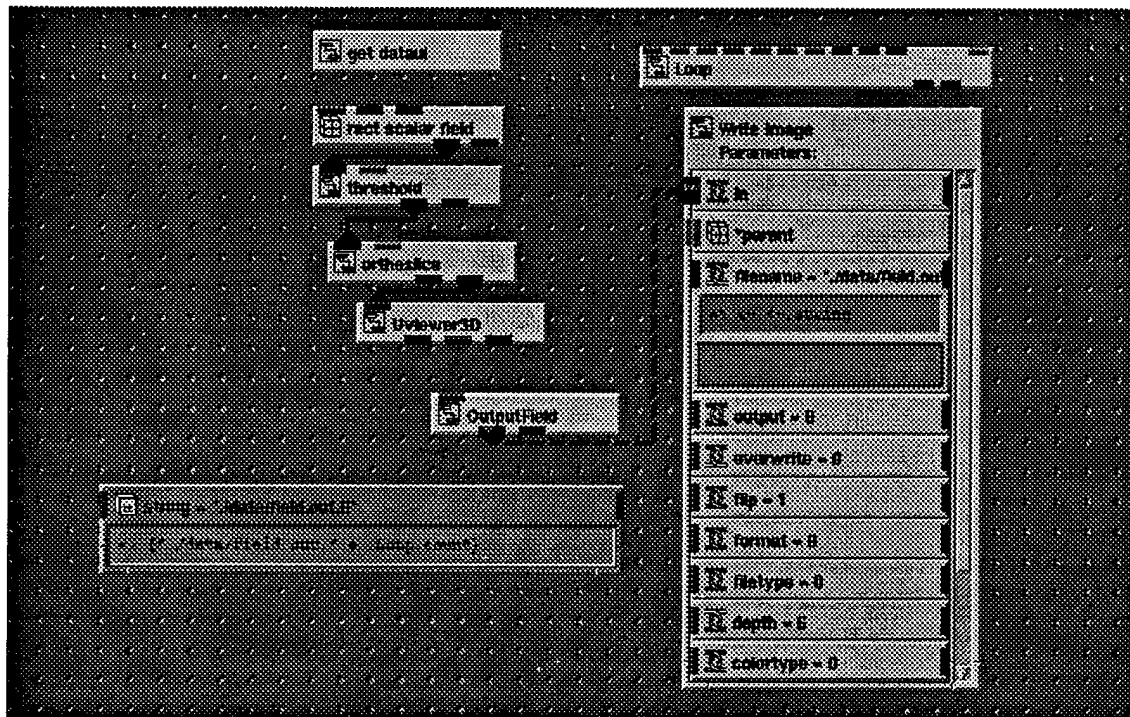


Figure 6: Write_Image plus string

a problem during the course of the animation (program crashing or spontaneous resizing of the image) then it is easy to return to the exact same settings.

Method 1

This method uses a Loop module and is appropriate for displaying data that change with time. An example *network* is shown in Figure 6. Under Main.IO is a module called Write_Image, which is exactly what it does. Its interface contains a slew of possibilities for the kind of image that can be exported. It does not automatically change the necessary options, like Bits per Pixel when the image type is changed. It also needs a filename, which is where AVS tricks become necessary. There may be another way to do this,

but this works. A string from `StandardObjects.parameters` will change the filename as the loop increments. This syntax is very important. (See Figure 6)

```
=> ("/path/name." + .Loop.count)
```

This will access the count variable and attach it to the end of the file name with each iteration of the loop. Now the `Write_Image` module must be changed to read that string. Right clicking on the `Write_Image` module will bring up a menu. `Parameters` is the least confusing choice (see Figure 5). This displays (simply) what is inside the module. There will be a link (with an icon like a knot) called "filename". Opening filename will reveal a reference to something within `Write_Image`. Delete it. In order to get its value from the string parameter, the reference should be changed to:

```
<-.<-.string.
```

That tells filename that it has to go up two levels and find an object named `string` to find its value. Hitting return inside the box will accept that value or bring up an error message. If an error message appears, adding another arrow, hitting return, and then deleting the arrow should work.

One noticeable thing about this *network* is that the `Write_Image` module will not connect to the `UViewer`. This suggests that another module is required to connect them. Under `Main.Viewers` is a module called `OutputField`. It has a purple input that matches one of the outputs from the `UViewer`, and a black, blue, and green striped output that matches the `Write_Image` input. The `OutputField` module outputs an AVS *field* of the image that the `UViewer` displays, and the `Write_Image` module turns it into a picture. Once everything is absolutely perfect, and not going to break and leave random files all over the harddrive, the "dynamic" option under `OutputField` should be toggled. This will update the *field* every time the loop increments and causes `Write_Image` to output a file. This process will create a numbered series of images (gif, tif, or other) which can then be catenated by another program (e.g. `Mediaconvert` on the SGI) to create an animation.

Method 2

The other method, which came built in with release 3.4 of AVS/EXPRESS is the animator and its partner, `image_capture`. They do exactly what their names imply, animate something and then capture it. `Image_capture` will also generate a movie after it creates a series of image files in AVS format. These modules were intended to smoothly capture actions within the viewer window such as resizing, translating,

or fading an object. The animator module requires something in the viewer to change for each frame.

The animator module does not require any hookups. It works by allowing the user to create a series of key frames. Once each frame is correct, click the generate frame button. If nothing has changed, it will not generate the frame. It will generate the frame, however, if something has moved and then moved back to its original position. This can become time consuming, but if the movie has to be done via these two modules, it can be done. The animator interface has a toggle that puts all of its interface into a separate window so that another interface can be open at the same time. The Transform interface (under the editors menu) can change the positions of things numerically so that they can be changed back more easily. After all of the frames are generated, click the play button to see if everything went correctly. If it did not, it has to be done again.

The image_capture module plugs into the purple output on the UViewer. The module is normally inactive. The name of the directory where the files will be placed should be entered in the movie name field (include the final slash). With this module there is no choice, the files come out as avs files with an extension of .x. These are for some reason colored differently then the image in the viewer so some experimentation is required to get the desired colors in the movie. Toggle the capture mode from disk to memory to disk again to outsmart a known bug. To capture the animation, the mode needs to be set to capture from view. If anything obscures the view while it is running, it will not be happy. The next step is to run the animation and hopefully the files will be generated. If so, the button that generates a movie can be pressed. It will leave an MPEG called anim.mpg in the directory with the images. This can also produce single images. Whenever it is set to capture from view, it will create images of every change in the viewer.

Streamlines

Those are the majority of the modules that were developed at the Center for Plasma Theory and Computation, but there are many other visualization possibilities. For example, streamlines can be created using the same macro that gathered and indexed data for the orthoslice. Streamlines requires vector data. The data that comes from the Fill up array module comes in the form of what should be vector data, but in a 1D array. That array can be fed through a float that is forced with the object editor to be $[n_x \times n_y \times n_z][3]$, and have a three element vector array which may then be used to create a field.

A few words on syntax

If a parameter (int, float, string) is going to be referencing something else, it needs a reference arrow (\Rightarrow). If it is referencing a sister object: one on the same level, or one on a lower level, the path consists of a dot and the object name, which can be very long if the object is nested. If it is referencing a higher object, it needs a directional (up) arrow and a dot ($\leftarrow\cdot$) for every level it needs to go.

From inside an object, a reference still needs a referencing arrow, but also needs as many arrows to get outside of the object it is nested in. Levels are separated by dots.

Values can be added by containing the string in quotation marks and adding the value reference with a "+".

Conclusion

In conclusion, the modules developed will assist the CPTC team by allowing them to create visuals and movies quickly and fairly painlessly with a program that can be very difficult to use. They now have the capability to animate 2D plots as well as incorporate them into movies. Electric and magnetic field data from ICEPIC may be read in without modifications and can then be used with all of the AVS modules after being converted to *field* format with the provided modules. This greatly increases the possible uses of these data. These modules were combined, along with ICEPIC data to create a movie which was shown at the AFOSR Mathematics and Computer Science Directorate Annual Review. That application also provides a template for future movies that can be made completely through the interfaces. In the near future, the computationalists at CPTC will, hopefully, use these modules so that they can be improved and eventually developed as stand-alone applications.

Acknowledgements

This work was performed in conjunction with work funded in part by the Air Force Office of Scientific Research Mathematics and Computer Science Directorate and the DoD High Performance Computing Modernization Program. AVS/EXPRESS is a product of AVS, Inc. ICEPIC was developed exclusively at the Center for Plasma Theory and Computation. Thanks to Kent Eschenberg and associates at CEWES MSRC for assistance and the use of their AVS modules. And special thanks to Captain Jerry Sasser.

References

[1] AVS/EXPRESS: User's Guide. Advanced Visual Systems Inc. Part number 320-0321-04.

<http://www.avs.com>.

[2] James J. Havranek and Bradley J. Smith, "A Portable Parallel Particle in Cell Code," AIAA paper#96-0835 (Jan.1996).

[3] Raymond W. Lemke, et al, "Investigation of a Load-Limited, Magnetically Insulated Transmission Line Oscillator (MILO)," *IEEE Transactions on Plasma Science*, **25**, p. 364 (1997).

DISTRIBUTION LIST

AUL/LSE Bldg 1405 - 600 Chennault Circle Maxwell AFB, AL 36112-6424	1 cy
DTIC/OCP 8725 John J. Kingman Rd, Suite 0944 Ft Belvoir, VA 22060-6218	2 cys
AFSAA/SAI 1580 Air Force Pentagon Washington, DC 20330-1580	1 cy
AFRL/PSTL Kirtland AFB, NM 87117-5776	2 cys
AFRL/PSTP Kirtland AFB, NM 87117-5776	1 cy
Official Record Copy AFRL/DEHE/Capt Gerald Sasser	10 cys